Learning a hidden Markov model-based hyper-heuristic

Willem Van Onsem, Bart Demoen, and Patrick De Causmaecker

KU Leuven
Department of Computer Science
Celestijnenlaan 200A, 3001 Heverlee

Abstract. A simple model shows how a reasonable update scheme for the probability vector by which a hyper-heuristic chooses the next heuristic leads to neglecting useful mutation heuristics. Empirical evidence supports this on the MaxSat, TravelingSalesman, PermutationFlowshop and VehicleRoutingProblem problems. A new approach to hyper-heuristics is proposed that addresses this problem by modeling and learning hyper-heuristics by means of a hidden Markov Model. Experiments show that this is a feasible and promising approach.

1 Introduction

A *hyper-heuristic* is a problem-independent algorithm that aims to select which heuristic to apply next during an evolutionary process. The aim of the hyper-heuristic is to speed up convergence toward an optimum in an optimization problem.

A hyper-heuristic is thus an optimization problem itself: one aims to optimize the convergence speed by scheduling heuristics appropriately. By problem independent we mean the hyper-heuristic can observe only some properties of the solution, and not the solution itself. Most software packages only allow inspecting the fitness-value.

One traditional approach to hyper-heuristics is to reward a heuristic that has - in the past - improved the solution, and punish the heuristics that computed a worse solution. This is achieved by adapting the probabilities by which a heuristic is chosen. We show with a simple model that such a scheme is doomed to underuse the *bad* heuristics, while they are necessary to find an optimal solution. This slows down convergence to an optimal solution. We report on experiments on four problem classes that confirm this.

Literature proposes several alternatives to avoid the underuse of certain heuristics in a more or less ad-hoc way. We choose for a more radical approach: we propose to model the choices made by a hyper-heuristic as a hidden Markov Model (HMM), and learn this HMM by means of (a sample of) the performance of the individual heuristics on (a sample of) the solution space.

This paper is structured as follows: Section 2 defines necessary terminology and concepts. In Section 3 we argue why a popular model – the probability vector

– will probably fail to increase convergence. This is done both using a model and empirically. Section 4 introduces our approach to modeling and learning hyper-heuristics: it is based on the Mealy Input-Output hidden Markov model (MIOHMM) also explained there. Results and experiments using this model are reported in Section 5. Section 6 concludes and discusses future work.

2 Preliminaries

A single-objective optimization problem consists of an implicit solution space $\mathcal{S}$ and a fitness function $f : \mathcal{S} \to \mathbb{R}$. The aim is to find a (pseudo) optimal solution $s^\star$ such that the fitness value $f(s^\star)$ is the infimum of $f(\mathcal{S})$.

An evolutionary algorithm aims to achieve this by applying a chain of heuristics on an initial solution s_0 and returns the best solution encountered so far when the time limit is reached.

A heuristic h is a function $h : \mathcal{S} \to \mathcal{S}$ that maps one solution to another solution.

Most heuristics are probabilistic in nature: the generated solution depends on both a solution and the seed of a random number generator. The set of possible outcomes of the heuristic h given the solution s, is called the *neighborhood* H of s, $H(s)$.

The concept of a heuristic can be generalized further: genetic algorithms for instance make use of heuristics that take as input two or more solutions. Such heuristics are called *"crossover" heuristics*. We do not consider such heuristics here.

We consider two types of heuristics: *local search* and *mutation heuristics*. A local search heuristic or *hill climber* is a heuristic h that guarantees that the generated solution is at least as good as the original solution, or more formally $f(h(s)) \geq f(s)$ for all solutions s. Mutation heuristics do not guarantee this behavior.

A *local optimum* with respect to a set of heuristics h_i is a solution s such that for each element s' in the union of the neighborhoods H_i of s, $f(s') < f(s)$ or $s = s'$. Any local search heuristic applied on a local optimum results in the same solution.

3 Modeling heuristic behavior with probability vectors

Hyper-heuristics[1,2] commonly use a *probability vector*[3] for guiding the selection of the heuristic to apply next.

A probability vector is a list of probabilities that sum up to one and associates elements - in this case heuristics - with probabilities. The well known *roulette wheel selection procedure*[4] can select heuristics proportional to their probability.

A probability vector is trained by updating the weights in function of accumulated empirical evidence. An algorithm that updates the probabilities is called an *update scheme*.

Hyper-heuristic systems use different [1,2] update schemes. We will show that under assumptions stated later, a *reasonable* update scheme eventually makes escaping from a local optimum less probable.

A reasonable update scheme rewards heuristics that produce a better solution, penalizes heuristics that produce a worse solution, and rewards or is neutral to heuristics that produce a new solution with the same quality. The update weights are furthermore monotonic with respect to the absolute difference in fitness value: if the difference increases, the weight either increases or remains the same. A reasonable update scheme is also oblivious to the type of heuristic: e.g. the update strategy does not differ between local search and mutation heuristics. Not all update schemes proposed in literature are reasonable.

3.1 On the probability to escape from a local optimum

Our claim is that probability vectors eventually antagonize convergence of the evolutionary process given. This is true under a number of reasonable assumptions.

1. The hyper-heuristics runs with both a local search and mutation heuristic: this is true for all hyper-heuristics we are aware off.
2. The heuristics are *stationary*. A heuristic is stationary if the probability of generating a solution only depends on the given current solution and not on other parameters like the elapsed time in the process.
3. It is very unlikely that the result of a mutation heuristic is a local optimum or that a mutation heuristic can improve the result of a local search heuristic. This seems true in practice. For simplicity, we assume here that it is not just very unlikely, but impossible.
4. For a mutation heuristic, the average fitness value of the solution after the application of the mutation heuristic is eventually worse than the fitness value of the solution before the application of the mutation heuristic.
   Experiments on four different problems[1] implemented in the *HyFlex 1.0* [5] framework show that this is true for 12 out of 15 mutation heuristics. Other mutation heuristics are iso-fitting: they produce always solutions that have the same fitness value as the original solution.

Once the evolutionary process is at a local optimum, a local search heuristic cannot generate a solution different from the active solution. Mutation heuristics are thus necessary to escape from a local optimum (assumption 1). Empirical evidence shows that evolutionary algorithms are stuck in a local optimum a significant number of times[2]. The detection of and the escape from local optima should thus be performed as efficiently as possible.

[1] See `goo.gl/vVTZNE` for details.

[2] If the heuristics are applied with uniform probability, around 5% to 20% of the time, although it strongly depends on the problem. See `goo.gl/vVTZNE` for empirical evidence.

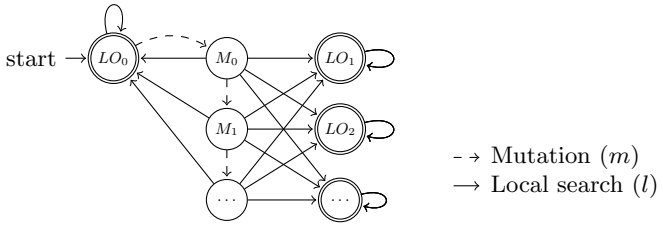

Fig. 1. Representation of the different states in an escaping process.

Figure 1 illustrates with an (in)finite state machine how a generic hyper-heuristic process escapes a local optimum. The nodes represent the possible states of a hyper-heuristic algorithm: they are distributions over the solution space $\mathcal{S}$ that represent how probable it is for a solution to be the "active solution". The escaping process starts in a state we represented by LO_0: the state represents the fact that a local optimum is the active solution at that moment. The aim is to get the system in another local optimal solution: other local optima are represented by the states $LO_1, LO_2, \ldots$. All local optima states are marked as "accepting" since they mark the end of an "escaping attempt".

In the initial state of the escaping process, the probability vector is represented by $\langle p_l, p_m \rangle$ with p_l and $p_m = 1 - p_l$ respectively the probability of the local search and mutation heuristic. The expected number of function calls before the mutation heuristic is called is determined by:

$$\sum_{i=0} p_l^i = \frac{1}{1 - p_l} = \frac{1}{p_m}. \tag{1}$$

During these escape attempts local search always generates the same solution: a reasonable update scheme either does nothing with this information or it rewards the local search heuristic l.

If an acceptance scheme is incorporated, this can take even more attempts since rejecting the result of a local search application makes no difference, and rejecting the solution generated by the mutation heuristic only results in more attempts to escape the optimum.

After the mutation heuristic m is eventually applied, the generated solution is worse since otherwise the initial solution would not have been a local optimum. Reasonable update schemes will penalize this with a reduction of the probability.

Since we assume it is impossible that the mutation heuristic produces a local optimum (assumption 3), the process escapes local optimum LO_0 and ends up in state M_0 at the cost of a decrease in the probability of the mutation heuristic. M_0 describes a probability distribution over the possible outcomes of the mutation heuristic. Now two possible scenarios can unfold:

1. The mutation heuristic is called a second time. The process ends up in state M_1 (with a possibly different distribution over the solution space) note that this scenario can be repeated;

2. The local search heuristic is applied and the system ends up in a (possibly different) local optimum. In case we end up in the same local optimum, all invested computational resources are wasted.

In the first scenario, in general the expected average fitness value over the distribution of solutions in M_{i+1} will be worse than that of M_i (assumption 4). Depending on the outcome of the mutation heuristic, we thus expect that the probability for the mutation heuristic will decrease further (since the probability vector is "reasonable"). After application of the mutation heuristic, the algorithm is still in an M-state and thus the two scenarios reemerge.

In the second scenario we reach a local optimum (assumption 3). The probability of the local search heuristic will increase at the expense of the probability of the mutation heuristic, since it is expected that the probability of the local search heuristic will increase. Equation (1) shows that the higher the probability of the local search heuristic, the longer it takes to escape a local optima.

One can describe this phenomena as the fact that the local search heuristic "takes full credit" for the work that was partially carried out by the mutation heuristic: escaping out of a local optimum. Since mutation heuristics are crucial in such process, at least a small probability should be maintained to prevent a hyper-heuristic locking itself in.

As the probability of applying a mutation heuristic decreases, it takes longer to escape from a (new) local optimum. Hence we claim:

Claim. A method using a probability vector with a reasonable update scheme eventually takes more and more time to escape from local optima and thus its convergence speed decreases.

3.2 Empirical evidence of the claim

We performed experiments using the *HyFlex 1.0*[5] framework to test whether our claim about probability vectors hold. Note that we proved our claim using just one mutation heuristic and one local search heuristic. In our experiments we used multiple heuristics of both kinds. Several reasonable update schemes were used. The tests were performed on the MaxSat, PermutationFlowshop, TravelingSalesman and VehicleRoutingProblem problems.

Figure 2 depicts the state of the probability vector at each generation for the VehicleRoutingProblem problem using a constant penalty/reward update scheme.

The thin gray line shows the fitness value of the active solution[3] and thus indicates whether the system is stuck in a local optimum. When the spikes in the fitness value are close together, this indicates that the escape from a local optimum was fast. When the fitness value remains the same during some generations, it indicates a slower escape. It is clear that as the number of generations increases, it becomes harder to escape the local optimum, still the fitness function $f(g)$ does not show that the local optima become significantly better.

[3] See the right axis for the appropriate unit.

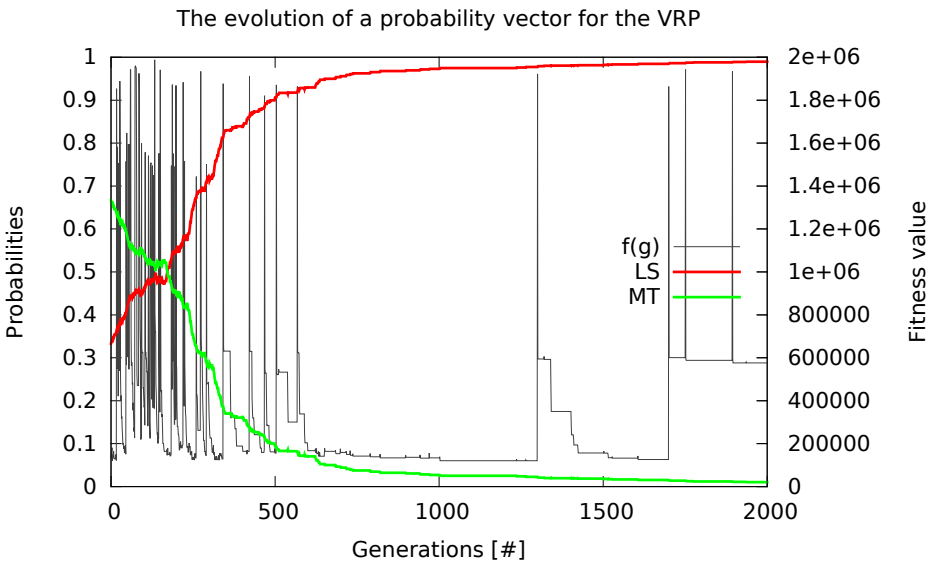

Fig. 2. Evolution of a probability vector solving the VehicleRoutingProblem.

This trend is matched by the evolution of the probability of the mutation heuristic, and consequently the local search heuristic: the thick red line indicates the total probability of the three employed local search heuristics. Initially the probability is set to 1/9 for each heuristic, so the total probability for the local search heuristics is $p_l = 0.333$. As the number of iterations grows, the probability approaches 1 quickly. The thick green line shows the sum of the probabilities of mutation and ruin-recreate heuristics. Since the problem runs with 6 such heuristics, initially the probability is set to $p_m = 0.666$, but it decreases fast below any reasonable probability.

The same effects were observed for all other tested problems and employed update schemes, so we conclude that our claim about probability vectors generally holds.

One can argue that in practice, hyper-heuristics never implement such a "pure" probability vector with a reasonable update scheme. For instance, many approaches use a probability vector per heuristic. This probability vector then determines which heuristic to apply next given the previous heuristic that was called first. We have performed empirical tests on such probability transition matrices as well and the same effects were observed although the convergence of the probability of local search heuristics towards 1 was slower. The reason seems that it takes several generations to update all the elements of the transition matrix.

3.3 Working around the problem with probability vectors

Hyper-heuristics try to solve the above stated problem in various ways:

Reinforcement learning[6,7] is a state-oriented update scheme with *memory*. It gives credit not only to the item last applied in the sequence, but uses a smoothing off approach where each heuristic in the sequence receives credit: the more recent the heuristic was called, the more the heuristic is rewarded. The rewards are given in the context of an implementation-defined state. A first limitation to this approach is that a programmer must find a good way to define states that can only depend on the observed fitness values. Furthermore reinforcement learning sometimes tend to reward items in a sequence that have nothing to do with the result: if multiple local search heuristics were applied in the evolutionary chain, they are all rewarded for delivering the same solution.

AdapHH[8,9] solves the problem using a *tabu search*[10] approach where heuristics that take a significant amount of time without generating a better solution are tabued. Since local search heuristics take in general more time than mutation heuristics, local search heuristics will get tabued more often. Mutation heuristics can get tabued as well resulting in a potential lower convergence rate. Since eventually the heuristics are untabued again, such algorithms have a more stable performance.

Finally, *Iterative Local Search*[11] interleaves mutation heuristics with local search heuristics and applies pairs of a mutation and local search heuristic. If such move generates a better solution, both heuristics are rewarded. A potential pitfall with this approach is that it can take more than one mutation heuristic application to get out of a local optimum. Extensions on iterative local search exist that take this into account.

We think that it might also be worthwhile to experiment with an update scheme that penalizes heuristics that generate the same (quality) solution as the given one. As a result, in a long sequence of local search heuristics, these local search heuristics will become less favorable and the evolutionary process can escape from the local optimum.

However, we think that it might be better to abandon probability vectors altogether.

4 From probability vectors to hidden Markov models

Although in the previous section we showed that a probability vector cannot learn well heuristic behavior, we think that probabilistic reasoning is a promising way to reason about heuristic behavior. The missing aspect in many implementations is maintaining a "state". This state is incorporated in a hyper-heuristic by the notion of the active solution.

We already discussed that reinforcement learning maintains states, but it is up to the programmer to decide what these states represent. In this section we discuss an approach were the semantic interpretation of states is left to a learning algorithm. This makes the algorithm more flexible.

We first discuss hidden Markov models and their application to hyper-heuristics in Section 4.1. We then show how we can learn heuristic behavior using such models in Section 4.2. In Section 4.3 and Section 4.4, we propose methods that

decide which heuristic to apply next in an evolutionary process and forgetting learned behavior in favor of new experience.

4.1 Hidden Markov models

Definition 1 (Hidden Markov model (HMM)). *A* hidden Markov model *is a* 3*-tuple* $\langle \boldsymbol{\pi}, A, B \rangle$ *with* $\boldsymbol{\pi}$ *a probability* n*-vector,* A *an* $n \times n$ *transition matrix and* B *an* $n \times m$ *emission matrix. Each row of* A *and* B *are probability vectors.*

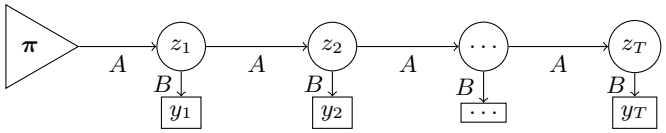

Fig. 3. A Markov process described by a hidden Markov model.

A hidden Markov model describes a Markov process as depicted on Figure 3: a probabilistic function that varies in time $Y : \mathbb{N} \to O : t \mapsto Y(t)$ with $O = \{o_1, o_2, \ldots, o_m\}$ a finite set of possible *observations*. This is done by considering a set of *"hidden" states* $\{s_1, s_2, \ldots, s_n\}$. Given the system is in state $z_t = s_i$ at time step t, $a_{i\,j}$ describes the probability of the system being in state $z_{t+1} = s_j$ at time step $t+1$. The hidden states cannot be observed directly. At each time step t the active state z_t emits an observation. The probability of emitting observation o_k is defined by $b_{i\,k}$. The initial probability vector $\boldsymbol{\pi}$ element for index i is defined as the probability of generating the corresponding solution as initial solution in the evolutionary process.

For the purpose of this paper, we take as observations the set of fitness values of the solutions. Since a hyper-heuristic can only inspect directly the fitness value of a solution, this is our only possibility.

Depending on which "behavior" we want to model, we can generate a set of observations O. Since the number of solutions in a combinatorial optimization problem is finite, the domain of behavioral aspects attached to the solution space is finite as well. From a hyper-heuristic point of view, the only reasonable behavioral property we can extract from a solution is its fitness value. Given the set of all possible fitness values O, the emission probability $b_{i\,k}$ is set to 1 if the solution corresponding to hidden state s_i has the fitness value represented by o_k and 0 otherwise.

The above discussed model shows that we can model the behavior of a single stationary heuristic with a HMM. The aim of a hyper-heuristic however is to determine which heuristic to apply next in a set of *multiple* heuristics. In order to model multiple heuristics, we use the concept of an Input-Output hidden Markov model[12].

Definition 2 (Input-Output hidden Markov model (IOHMM)). *An* Input-Output hidden Markov model *is a* 3*-tuple* $\langle \boldsymbol{\pi}, A, B \rangle$ *with* $\boldsymbol{\pi}$ *a probability* n*-vector,* A *an* $l \times n \times n$ *transition matrix and* B *an* $n \times m$ *emission matrix. Each matrix* A_i *is a transition matrix as defined in Definition 1.*

An IOHMM considers not only a set of observations O but an input alphabet Σ as well. In the case of a hyper-heuristic the alphabet consists out of the set of heuristics one can apply. Depending on the input h_i, a different transition matrix A_i is applied. One can generate an IOHMM model for a set of heuristics H and an initializer analogue to a HMM, but the process of calculating the transition matrices A_i is repeated for each heuristic.

4.2 Learning heuristic behavior

Constructing an IOHMM for a specific problem instance is useless: first of all it requires at least $\mathcal{O}\left(|\mathcal{H}| \cdot |\mathcal{S}|^2\right)$ time, with $|\mathcal{H}|$ the number of heuristics and $|\mathcal{S}|$ the number of solutions of the problem instance, to generate an IOHMM for a problem instance. Thus, it is easier to enumerate the entire solution space in search for the global optimum.

An evolutionary process generates empirical evidence: a list of tuples containing both the heuristic that was called and the resulting behavior (i.e. the fitness value of the generated solution). The well known Baum-Welch algorithm uses the *Expectation-Maximization* methodology to learn values for $\boldsymbol{\pi}$, A and B such that probability of generating a sequence like the empirical evidence is maximized for an a priori determined number of hidden states n. This is the best we can hope given we cannot make any assumptions regarding how the heuristics work.

The algorithm runs in $\mathcal{O}\left(t \cdot \left(n^2 + n \cdot m\right)\right)$ with t the number of data points, n the number of hidden states and m the number of possible observations. This is the time complexity of one step in the expectation-minimization process: it is possible that multiple iterations are necessary before the model parameters $\langle \boldsymbol{\pi}, A, B \rangle$ converge towards a local optimum[4]. At this point we realized that the number of observations and hidden states is too large to learn a model effectively, so they must be reduced.

By reducing the number of hidden states, the hidden states no longer represent solutions, but distributions over the set of solutions. Each distribution marks solutions that show, according to the Baum-Welch algorithm, similarly with respect to the observations. Since the algorithm aims to maximize the probability of the observed data, solutions will be grouped if one or more heuristics behave similarly on both solutions. The number of hidden states can be a limiting factor: if the heuristic behavior is complex, it requires more hidden states. As the amount of empirical evidence grows, one can increase the number of hidden states to increase the quality of the model.

[4] Experiments show that such local optimum is nearly always near the global optimum, although sequences can be derived that are hard to learn.

We also reduced the set of observations (in the context of a hyper-heuristic, the set of possible fitness values). If an IOHMM considers the set of all possible fitness values, the model has no means to generalize heuristic behavior and the model would have a hard time learning that a local search heuristic applied to one local optimum would generate the same solution, regardless of the fitness value of the first solution.

Reasoning about the difference between two fitness values is therefore a better decision: it enables the model to learn that if there is no difference between the initial and final solution of a local search heuristic, there never will be any in the future.

Considering "differences" between two fitness values as the observation set leads to an inconsistency: differences between fitness values of two solutions do not correspond to a single solution. Heuristics can produce different fitness differences for the same solution.

We can solve this issue by squaring the number of hidden states: in that case each hidden state represents a tuple containing the old and the new solution. In that case the hidden Markov model has a "memory"[5] of 1 time step. The computational effort invested in learning how to handle such memory will however increase significantly: the number of parameters to learn is now $n^4 + n^2 \cdot m$ with n the number of original hidden states and m the number of "difference" observations.

One can "pre-encode" the use of memory using a *Mealy Input-Output hidden Markov Model*: a IOHMM where the observed difference depends on both the solution and the heuristic applied on that solution.

Definition 3 (Mealy Input-Output hidden Markov model (MIOHMM)). *A* Mealy Input-Output hidden Markov model *is a* 3*-tuple* $\langle \boldsymbol{\pi}, A, B \rangle$ *with* $\boldsymbol{\pi}$ *a probability* n*-vector,* A *an* $l \times n \times n$ *transition matrix and* B *an* $l \times n \times m$ *emission matrix. Each matrix* A_i *is a transition matrix and each matrix* B_i *is an emission matrix as defined in Definition 1.*

The observation y_t no longer depends on current hidden state z_t, but on the previous hidden state z_{t-1} and the input token x_t. The Baum-Welch algorithm can be modified such that it trains a MIOHMM in the same time complexity as training a hidden Markov model. Figure 4 depicts the evolution of a MIOHMM in time.

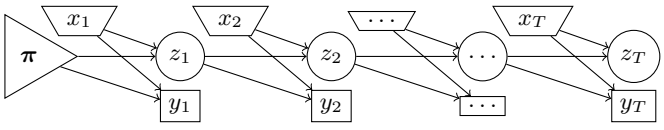

Fig. 4. The Markov process described by a Mealy Input-Output hidden Markov model.

[5] This in contrast with the *Markov assumption* that states that a Markov process has no memory.

For our experiments, we reduced the number of observations to three: *better*, *worse* and *same*.

4.3 Selecting a heuristic

Based on the information collected, compressed and stored by the MIOHMM, one needs to decide which heuristic to apply next in the evolutionary process. An advantage of a hidden Markov model is that it can calculate the distribution over the hidden states at any point in the process. Based on the earlier historical evidence and the model itself, one can predict with which probability a heuristic will produce a better, equivalent or worse solution, this of course given the model is correct.

Our hyper-heuristic submitted to the *CheSC 2014*[6] challenge used the following selection procedure: we designed a desired emission probability vector with values:

$$\boldsymbol{d} = \langle d_+, d_=, d_- \rangle = \langle 0.6, 0.05, 0.35 \rangle \tag{2}$$

At each decision point the heuristic behavior is predicted. The heuristic for which the dot-product between $\boldsymbol{d}$ and the predicted output is maximized is selected as the next heuristic.

We did not perform any tuning on the $\boldsymbol{d}$ vector: the vector merely favors generating a better solution over a worse solution and a worse solution over an equivalent solution. Since we are only interested in the heuristic that maximizes the dot-product we think this is a robust metric: a small difference in the $\boldsymbol{d}$-vector will typically only lead to a different decision on rare occasions.

The selection procedure is still a weak spot in our hyper-heuristic algorithm.

4.4 Forgetting learned experience

Since the number of empirical samples keeps increasing, adapting our model to the latest measuring point would require increasing computational effort at each time step. By considering only a time window of samples, we set a threshold on the maximum amount of effort spend on improving the learned model.

Since the observed data originates in many cases from the same region in the search space, the MIOHMM will learn a model aligned to this region. By considering a time frame, our algorithm has the ability to forget past experience that would make the model less suited for the challenges for the evolutionary process at that moment.

The Baum-Welch algorithm tends to stick with an earlier learned model. For instance, transition probabilities close to 0.0 require many iterations to increase to a significant level. Since the transition matrices of learned hidden Markov models tend to be sparse learning a better model can be hard.

We solved this by adding additional noise to the matrices: small probabilities were added to or subtracted from the elements from transition matrices. This

[6] See Section 5.3.

noise can be seen as "forgetting" what has been learned in favor of accepting new experience. *Markovitch*[13] argues that forgetting is a vital point in learning that many algorithms tend to ignore.

5 Results

In this section we show that local search heuristics can be learned perfectly, the effect of the number of hidden states on the model quality and the hyper-heuristic performance in practice.

5.1 Local search heuristics

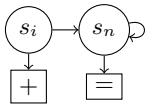

Fig. 5. A model of a local search heuristic requires two hidden states.

An encouraging theoretical result of the use of hidden Markov models, is that the model can easily learn the behavior of local search heuristics using two hidden states. The two states are called the *improvement state* s_i and the *non-improvement state* s_n.

The probability of a better solution (+) in the first state is 100% as is the probability of generating an equal solution (=) in the second. The transition probability of s_n to itself is $p_{nn} = 100\%$ as well since our local search heuristic reached an optimum.

In case the *local search heuristic* guarantees a local optimum after one function application (as is the case in *HyFlex 1.0*[5]), the transition from the improvement state s_i to the optimum state is 100%. In case it can take an undetermined number of applications of the heuristic, the probability is $p_{in} = 1/\lambda$ with λ the average number of consecutive improvements until an optimum is reached. The transition probability from the improvement state to itself is defined by $p_{ii} \equiv 1 - p_{in}$.

If the result is guaranteed to be a local optimum, this behavior can be learned from three observations. Otherwise it requires a sequence of heuristic applications until an optimum is reached to estimate p_{in} effectively. The precision of p_{in} increases with $\mathcal{O}\left(1/\sqrt{k}\right)$ with k the number of sequences of the local search heuristic that end up in a local optimum.

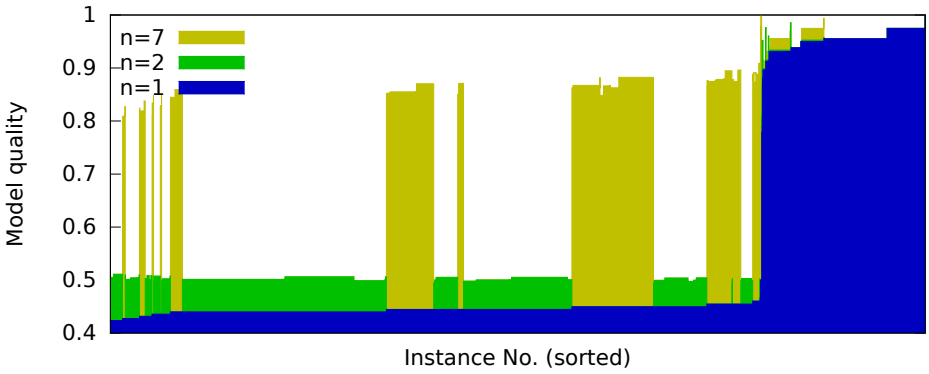

Fig. 6. The effect of the number of hidden states on the model quality.

5.2 Number of hidden states versus model quality

An advantage of the hidden Markov model approach is that the learning component acts rather independent from the decision component[7]. This allows one to analyze whether the MIOHMM is capable of modeling the heuristic behavior correctly. Sometimes machine learning algorithms fail to improve the overall model quality: some problem instances can be modeled better at the expense of others so that we end up with a zero-sum result.

We performed a batch of experiments in which we iterated over all possible Max3Sat problems with 8 variables and 4 clauses. With symmetry breaking, this results in 199′057 unique problems. For each problem, we tried to learn the behavior of three low level Max3Sat heuristics with a MIOHMM for a varying number of hidden states using data collected exhaustively over the entire solution space.

The quality of the learned model was evaluated by calculating the average number of times the model could predict the result of heuristic application in an evolutionary process correctly. Although we argued that the selection procedure is still a weak spot in our approach, the more accurate a model can predict the outcome of a heuristic, the better the decision a hyper-heuristic can make.

Figure 6 shows the results obtained with 1, 2 and 7 hidden states. As the number of hidden states increases, the achievable quality of a MIOHMM is guaranteed to increase[8]. Since there is no inherent order in Max3Sat problem instances, we ordered the problem instances on increasing model quality of a MIOHMM with 1 hidden state. The regions in green and yellow show the increase of model quality compared to a MIOHMM with less hidden states.

The graph shows that as the number of hidden states increases, the model quality of certain chunks of problem instances increases significantly. For some

[7] The decision component will however have an impact on the generated evidence.

[8] Since the Baum-Welch algorithm is a heuristic learning algorithm, this is not guaranteed, but we never encountered such an example.

instances, the learned model predicts the behavior correctly in more than 80% of the cases.

The results might seem not that impressive, but note that a completely random selection would result in a model quality of 0.333. Moreover in this experiment we aimed to learn the heuristic behavior over the entire search space.

Since this data is not available in a real hyper-heuristic process, the hyper-heuristic will learn based on evidence of "local" data and thus specialize in the active region.

To the best of our knowledge, this is the first experiment performed with hyper-heuristics where a problem instance set is exhaustively enumerated. These experiments are an indication that one hyper-heuristic is more suited than another to learn generic heuristic behavior given the learning algorithm can be separated from the decision algorithm. The full batch of experiments is available at `goo.gl/vVTZNE`.

5.3 Hyper-heuristic performance

Our hyper-heuristic based on MIOHMM was submitted to the *CheSC 2014* [14,15] competition for the parallel track. Parallelization was performed by multiple threads, learning and selecting based on the same model. No a priori knowledge about the heuristics[9] was added and the parameters as described by equation (2) were not fine-tuned.

The algorithm outperformed the only competitor, the *Evolving Tree Hyper-Heuristic (ETH)*[16] for the ProbeSelectionProblem and MultidimensionalKnapsackProblem problem, but achieved worse results for the VehicleRoutingProblem. For more details, visit `goo.gl/IQZ1bj`.

6 Conclusion and Future work

We have shown that the probability vector approach has problems learning heuristic behavior.

As an alternative, we have modeled a hyper-heuristic as an IOHMM. To effectively learn such model, its number of observations and hidden states must be reduced. This can result in a model with lower quality. Whether this is a serious issue depends on the problem at hand.

An additional advantage of our approach is that at each point in time, one can measure how well the model is trained with respect to historical data. The model can learn from multiple sources concurrently and thus has a benefit with respect to parallelization as was shown on the *CheSC 2014* competition.

On the whole we think our results are promising and warrant further investigation. In particular, further research must study alternatives for reducing the observation set. Also the selection procedure can be improved. It is possible to encode a priori known aspects about heuristics into our model such that well known aspects should not be learned.

[9] For instance, how a hyper-heuristic can model a local search heuristic.

Acknowledgements

This research is funded by the *Institute for Innovation through Science and Technology (IWT)* under grant 131′751.

References

1. I. Khamassi, "Ant-Q Hyper Heuristic Approach applied to the Cross- domain Heuristic Search Challenge problems," in *CHeSC 2011*, 2011.
2. K. McClymont and E. Keedwell, "A Single Objective Variant of the Online Selective Markov chain Hyper-heuristic (MCHH-S)," in *CHeSC 2011*, 2011.
3. Wikipedia, "Probability vector — wikipedia, the free encyclopedia," 2014. [Online; accessed 24-September-2014].
4. A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 6, pp. 2193 – 2196, 2012.
5. G. Ochoa, M. Hyde, T. Curtois, J. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. Parkes, S. Petrovic, and E. Burke, "HyFlex: A Benchmark Framework for Cross-domain Heuristic Search," *European Conference on Evolutionary Computation in Combinatorial Optimisation(EvoCOP 2012)*, vol. 7245, pp. 136–147, 2012.
6. C. J. C. H. Watkins, *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989.
7. L. D. Gaspero and T. Urli, "A Reinforcement Learning approach for the Cross-Domain Heuristic Search Challenge," in *CHeSC 2011*, 2011.
8. M. Mısır, P. De Causmaecker, G. Vanden Berghe, and K. Verbeeck, "An adaptive hyper-heuristic for CHeSC 2011," in *CHeSC 2011*, 2011.
9. M. Mısır, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe, "An Intelligent Hyper-Heuristic Framework for CHeSC 2011," in *Learning and Intelligent Optimization* (Y. Hamadi and M. Schoenauer, eds.), Lecture Notes in Computer Science, pp. 461–466, Springer Berlin Heidelberg, 2012.
10. F. Glover and S. Hanafi, "Tabu search and finite convergence.," *Discrete Applied Mathematics*, vol. 119, no. 1-2, pp. 3–36, 2002.
11. E. K. Burke, T. Curtois, M. R. Hyde, G. Kendall, G. Ochoa, S. Petrovic, J. A. V. Rodríguez, and M. Gendreau, "Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms.," in *IEEE Congress on Evolutionary Computation*, pp. 1–8, IEEE, 2010.
12. F. Bause, "Input-output hidden markov models for the aggregation of performance models," Sfb Teilprojekt M and Ls Informatik Iv and Modellierung Grosser and Sfb Teilprojekt M SFB559-03010, Universität Dortmund, Dortmund, juli 2003.
13. S. Markovitch and P. D. Scott, "The role of forgetting in learning.," in *ML* (J. E. Laird, ed.), pp. 459–465, Morgan Kaufmann, 1988.
14. S. Asta, E. Özcan, and A. J. Parkes, "Batched mode hyper-heuristics," in *Learning and Intelligent Optimization* (G. Nicosia and P. Pardalos, eds.), Lecture Notes in Computer Science, pp. 404–409, Springer Berlin Heidelberg, 2013.
15. W. Van Onsem, B. Demoen, and P. De Causmaecker, "HHaaHHM: Hyper-Heuristics as a Hidden Markov Model," in *Proceedings of the Cross-domain Heuristic Selection Competition 2014*, April 2014.
16. J. Pihera and N. Musliu, "ETHH - Evolving Tree Hyper-heuristic," in *Proceedings of the Cross-domain Heuristic Selection Competition 2014*, April 2014.